社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

コードクローンに対する変更の一貫性と欠陥発生との関連性に関する分析

西田　皓司†　　伏田　享平†　　川口　真司†　　飯田　元†

† 奈良先端科学技術大学院大学 情報科学研究科
E-mail: †{koji-ni,kyohei-f,kawaguti}@is.naist.jp, ††iida@itc.naist.jp

あらまし　ソースコード中に存在する重複コード列をコードクローン (以下，クローン) という．本稿では，互いにクローンの関係にあるコード片全てに一貫していない変更を行うことと，その後生じる欠陥との関連性を明らかにすることを目的として，クローンが開発の過程でどのように作成され改変されてきたのかを示すクローンの履歴情報を用いてそれら関連性を分析した．分析では，ソフトウェアのあるバージョンにおいて，互いにクローンの関係にあるコード片全てに一貫していない変更を行った場合に生じると考えられる,「クローンの分岐」と「バグの発生」との関連性についての仮説を設けた上で，仮説の実証を試みた．分析の結果，分岐を含まない場合と比べて，分岐を含む場合の方がバグが発生する傾向にあり，その数も多いことが分かった．この結果から，クローンの分岐は，その後のバグの発生に影響しており，そのようなクローンに対しては特に注意を払うことが有効であることを確認した．
キーワード　コードクローン，改版履歴情報，コードクローンの分岐，コードクローンの有害性，版管理システム

An Analysis of Relation Between Inconsistent Changes of Code Clone and Bug Occurrence

Koji NISHIDA†, Kyohei FUSHIDA†, Shinji KAWAGUCHI†, and Hajimu IIDA†

† Graduate School of Information Science, Nara Institute of Science and Technology
E-mail: †{koji-ni,kyohei-f,kawaguti}@is.naist.jp, ††iida@itc.naist.jp

Abstract　Duplicated code in the source code is called a code clone. In this paper, for revealing relation between inconsistent change to clone group and the occurrence of the bug, we analyzed code clone history which represents how code clone was changed in the development process. In the analysis, we made hypotheses of the relation between "divergence of the code clone", generated by inconsistent change to clone group, and "bug occurrence". Then, we carried out an experimentation using Eclipse development data. The result shows that code clone containing the divergence tends to have more bugs. From this result, we made a conclusion that divergence of the code clone influenced the bug occurrence, and we need to pay attention to such clone.
Key words　Code clone, Change history, Divergence of code clone, Harmfulness of code clone, Version management system

1. はじめに

近年，ソフトウェアの大規模化と運用の長期化に伴い，ソフトウェア開発工程における保守工程の占める割合は年々増加している．保守工程における大きな問題の 1 つとして，ソースコード中に含まれる重複したコード列であるクローンの存在が挙げられる．例えば，クローンを含むソースコードに不具合が見つかった場合，関係する他の全てのクローンについて，同様の修正が必要か調査した上で，必要な箇所には修正を施し，さらに修正箇所が正しく動作するかを再テストしなくてはならないためである．

このような背景から，ソースコード中のクローンを見つけ出し，積極的に除去すべきであるということが多く言われている．しかし，すでに存在するクローンを除去するための修正作業には多大なコストがかかる上に，新たなバグが混入し，逆にソフトウェアの信頼性の低下を招く恐れがある．また，熟練したプログラマーが意図的にクローンを生成するというような，クローンを肯定的に利用する場合もある [1]．つまり，全てのクローンが問題とは一概に言い切れないため，検出されたクローンの中から，除去すべき・注意すべきクローンだけを見極める

ことができなければ，クローンに対する保守作業の効率が低下してしまう．

本稿では，除去すべき・注意すべきクローンとして，「互いにクローンの関係にあるコード片全てに一貫した変更が行われずに一部のみ変更された」クローンに注目した．このような変更が行われると，元々1 種類のクローン群だったものが，2 種類の別々のクローン群として検出されるようになる．このとき，もしどちらか一方のクローン群にバグなどの修正すべき箇所が見つかった場合，もう片方のクローン群にも同じソースコードが多く含まれており，同じ修正を必要とする可能性が高くても，2 つのクローン群がすでに別種のクローン群として認識される．そのため，このような部分を特定することは，クローン検出ツールを利用しても発見が困難となる．

そこで，本研究では，クローンが開発の過程でどう変化してきたのかを示すクローンの履歴情報に着目することで，このようなクローンを判別し，有害性について調査することを目的とする．履歴情報には，バージョン間でのクローンの関係や，いつどのクローンに対して変更が行われたのか，いつどのクローンで問題が生じたのかといった様々な情報が含まれる．よって，履歴情報を用いることで，一貫した変更が行われなかったクローンを判別し，そのようなクローンが問題を引き起こしているのかを調査することができる．

このような一貫していない変更が問題を引き起こすことを確かめるために，履歴情報を用いて以下の仮説を実証する分析を行った．

「あるバージョンにおいて，互いにクローンの関係にあるコード片全てに一貫した変更を行わずに一部のみを変更した場合，その後，それらのコード片について，バグが発生しやすくなる.」

2. 関 連 研 究

Kim らは，クローンの履歴情報に着目した分析によって，クローンの分類を行っている [2]．分類の結果，長期間一貫した変更を加えられてきたクローングループや，言語の制約によって取り除くことができないクローンが多く存在し，これらはリファクタリングによって取り除くのが困難なクローンであるとしている．一方で，短期間しか存在しないクローンに対しては，リファクタリングの必要がないとしている．この研究は，履歴情報に着目しているという点では我々の研究と似ているが，リファクタリングの可否でクローンの分類を行っているという点で異なっている．

Krinke は，クローンに対する変更は一貫した変更であるという仮説のもと，5 つのプロジェクトに対して実験を行い，クローンに対する一貫した変更の割合について調査を行っている [3]．その結果，一貫した変更は 50 パーセント程度で，一般的に言われているよりも少なかったと結論付けている．この研究も我々の研究と同様，過去の履歴情報を用いた分析を行っているが，クローンが与えるソフトウェアへの影響については言及していない．

Juergens らは，クローンの存在が実際のソフトウェアに悪影響を及ぼしているのかを調べるために，5 つのプロジェクトに対してケーススタディを行った [4]．ケーススタディでは，クローンセット中のクローンに対して一貫した変更を行っていないケースがどの程度あるのか，さらにそれが意図したものなのかということについて分析を行っている．結果，一貫しない変更が多く存在し，それが意図しないものであった場合は，問題が発生しやすいということを示した．この研究は，クローンに対する変更の一貫性に着目し，クローンがソフトウェアに及ぼす悪影響について述べている点で，我々の研究と類似しているが，影響性に関する判断に際して，明確な基準を設けておらず，我々の研究よりも定性的な分析であるといえる．

左藤らは，クローンの長さ，広がり，複雑度という 3 つのメトリクスに基づき，クローンとソフトウェアの信頼性との関係について調査を行っている [5]．調査の結果，クローンの種類によって，信頼性を向上させる性質を持つクローンと信頼性を低下させるクローンに分類している．この研究は，定量的な分析により，クローンを評価しており，その点で我々の研究と共通している．しかし，評価の基準や，履歴情報に着目していないという点で我々の研究とは異なっている．

3. クローン・クローン履歴に関する諸定義

本節では，本稿での分析対象であるクローンの定義について述べる．そして，クローンの有害性を評価するために本稿が着目しているクローンの履歴，およびその取得方法について述べる．また，クローンへの一貫しない変更をモデル化した概念であるクローンの分岐について述べる．最後に本研究においてクローンの有害性の指標となる，クローンに関するバグ修正記録について述べる．

3.1 クローンの定義

クローンとはソースコード中の重複したコード列のことである．これはソフトウェアの開発中や保守作業中に，次の要因によって生じる [6].

- コード列のコピー＆ペースト
- コードジェネレータによるコード生成
- 特定のコーディングスタイルの利用
- 偶然の一致

どのようなコード列をクローンと見なすかの定義は，検出方法やツールごとに異なるが，本稿では，クローンの検出に CCFinderX [7] を用いるため，CCFinderX の定義に従う．CCFinderX では，変数名の違いや，改行とインデントの違いがある場合でもクローンとして検出することが可能である．

3.2 クローン履歴

過去に存在したクローンが現在のソースコードのどこに対応するのかという情報をクローン履歴という．クローン履歴とクローン履歴の必要性について，図 1 を用いて説明する．

図 1 は，バージョン V_i とバージョン V_{i+1} のそれぞれにおいて，検出されたクローンを表している．本研究では，a_1，a_2，a_3，a_4 それぞれを 1 つのクローンと呼ぶ．また，(a_1, a_2) のようにクローン検出ツールによって類似コードと判定されたクローンを，互いに「クローン関係」があると呼ぶ．そして，互

いにクローン関係にあるクローンの集合を「クローンセット」と呼ぶ．また，図中の点線は，クローン履歴を表しており，クローン履歴によって結ばれる 2 つのクローンを，お互いに「履歴関係」にあると呼ぶ．本研究では，クローン履歴および履歴関係の定義として，川口らが提案しているクローン履歴関係 [8] の定義を用いた．

次にクローン履歴の必要性について述べる．図 1 でバージョン V_i の時点でクローンセット α のクローン a_3，a_4 に対して変更を加えた結果，バージョン V_{i+1} では a_3，a_4 はクローンセット β となる．このとき，クローンセット α' とクローンセット β は関連性がある可能性が高いが，バージョン V_{i+1} において 2 つは別のクローンセットとして検出されてしまう．そのため，これ以後，一方で変更を加えた場合，もう一方のクローンセット内のクローンも変更する必要があるにもかかわらず，見逃してしまいやすくなってしまう．クローン履歴を用いることで，このような 2 つのクローンセットを関連性のあるものとして判別することが可能となる．

クローンの履歴関係を取得するためには，まず過去のソースコードが必要である．過去のソースコードを得るためには，版管理システムを用いる．版管理システムとは CVS [9] や Subversion [10] のようにプロダクトの保管・管理に用いられるシステムである．版管理システムでは，管理下のプロダクトを任意の時点の状態に復元して取得する機能が提供されている．

次に，版管理システムを用いて取得した過去のソースコードそれぞれに対して，CCFinderX を用いてクローンの検出を行い，その結果に対して川口らの手法 [8] によって，履歴情報を取得する．

3.3 クローンセットの分岐

3.2 で記述したように，クローンセット中の一部のクローンにだけ，変更を加えることによって，次のバージョンでは，クローンセットが 2 つ，もしくは 3 つ以上の独立したクローンセットとして存在することがある．本稿では，このような現象を,「分岐」と呼ぶ.

図 2 は，クローンセットが，変化・分岐する様子を示している．詳細を以下に記す.

- バージョン V_{i-1} において，クローン a_1，a_2 からなるクローンセット α が存在している.
- a_1，a_2 をコピー&ペーストすることにより，バージョン V_i では，a_1，a_2，a_3，a_4 からなるクローンセット α となる.
- a_3，a_4 に対して変更を加えることにより，バージョン V_{i+1} では，クローンセット α' とクローンセット β に分岐する.

また，本稿では，クローン関係および履歴関係によってお互いに到達可能なクローンの集合を，1 つの「クローンセットヒストリー」と定義する．図 2 の例では，クローン a_1，a_2，a_3，a_4，b_1，b_2 は 1 つのクローンセットヒストリーに属する.

3.4 クローンに関するバグ修正記録

本研究では，有害なクローンか否かを，クローン履歴の中にバグが修正された記録を含んでいるかどうか，で判断する.

3.2 節で述べたクローン履歴は，コミット（版管理システムの管理下にあるファイルを更新すること）されたときに記録されるコミット情報との対応関係を保持している．さらに，コミット情報の中にバグへの対処記録があるかどうかを判定するために，Fischer らの手法 [11] を用いる．Fischer らの手法は，版管理システムに加えて Bugzilla や GNATS などのバグ管理システムを利用しているソフトウェアの開発記録を対象に，コミット情報に残されたバグへの対処記録を抽出する手法である.

本研究では，Fischer らの手法によって，コミット情報からバグの対処履歴を取り出すことが可能なソフトウェア開発記録を対象として分析を行う.

4. 仮　　説

これまでに述べたように，クローンの分岐は後々問題を引き起こす可能性がある．このことからから「あるバージョンにおいて，互いにクローンの関係にあるコード片全てに一貫した変更を行わずに一部のコード片のみを変更した場合，すなわち，分岐が生じた場合，その後，それらのコード片について，バグが発生しやすくなる」という考えのもと，以下の 3 つの仮説を立てた.

仮説 1　分岐を含むクローンセットヒストリーの方が，分岐を含まないクローンセットヒストリーよりもバグ修正を含む割合が高い (図 3(1) 参照)

仮説 2　バグ修正を含むクローンセットヒストリーの方が，バグ修正を含まないクローンセットヒストリーよりも分岐を含む割合が高い (図 3(2) 参照)

仮説 3　分岐を含むクローンセットヒストリーの方が，分岐を含まないクローンセットヒストリーよりも，より多くのバグ修正を含む (図 3(3) 参照)

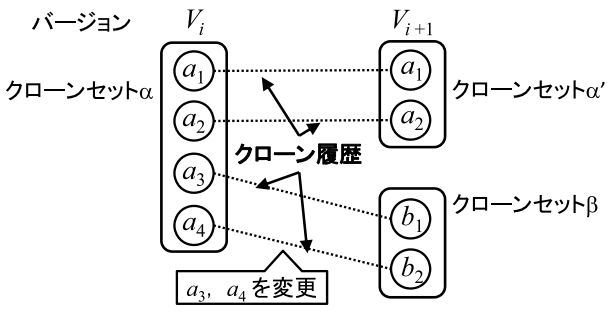

図 1　クローン履歴

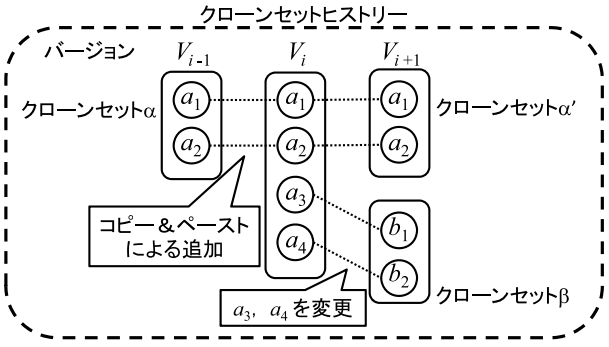

図 2　クローンの変化と分岐の例

そして，以上 3 つの仮説を実証するために，履歴情報を用いて分析を行った．

5. ケーススタディ

本節では，前節で示した仮説を実証するために，ケーススタディを行った．ケーススタディの手順，結果を以下に記す．

5.1 対象データ

オープンソースの統合ソフトウェア開発環境 Eclipse のバージョン 2.0，2.1，3.0 のソースコード，およびその間の編集履歴情報・バグ情報を対象とし，分析を行った．編集履歴情報とバグ情報の対応関係に関しては，Fischer らの手法に基づいて Eclipse の編集履歴情報・バグ情報を分析した結果である Zimmermann らによるデータ [12] を用いた．また，本実験では，一部のモジュール (org.eclipse.jdt.ui) を実験対象とした．

5.2 分析手順

5.2.1 クローンの検出と履歴情報の取得

トークンベースによるクローン検出手法を採用している CCFinderX [7] を用いて，それぞれのリビジョンでのソースコードからクローンを検出した．検出の際の設定におけるトークン数は，CCFinderX の初期値である 50 を採用した．

得られた結果から，3.2 節で述べた方法により履歴情報を抽出した．

5.2.2 クローンセットヒストリーの分類

クローンセットヒストリー中に，コミットを含むか，バグ修正のあったクローンを含むか，分岐を含むか，バグが分岐の後で発生しているか，の観点で，クローンセットヒストリーを分類し，以下の A, B, C, D, E, F の 6 グループに分類した．

A: バグ有り・分岐有り・分岐の後にバグ有り
B: バグ有り・分岐有り・分岐の後にバグ無し
C: バグ有り・分岐無し
D: バグ無し・分岐有り
E: バグ無し・分岐無し
F: コミット無し

以降，A グループのクローンセットヒストリーの総数を $|A|$ と表記する．B，C，D，E，F についても同様の表記を行う．

分類は，以下に示す手順 **(1)**，**(2)**，**(3)**，**(4)** の順で行った．また，図 4 に，分類手順を示す．

(1) 全てのクローンセットヒストリー群の中から，コミットのあったクローンを含むクローンセットヒストリーのみの抽出を行った．

(2) **(1)** で抽出されたクローンセットヒストリー群に対して，バグ修正を含むか含まないかで分類を行った．

(3) **(2)** で分類されたクローンセットヒストリー全てに対して，分岐を含むか含まないかでさらに分類を行った．

この分類の際，以下の 2 点の条件のいずれかを満たした場合に「分岐が有る」とした判断した．

条件 1: ある時点でクローンセットの数が 2 つ以上ある場合（図 2 の V_{i+1} のような状態）．

条件 2: クローンとしては認識されていないが，クローンの可能性が高い場合 (詳細は後述)．

条件 2 は，分岐の特別な場合として単一のコードクローンにのみ変更が加えられたケースを想定した条件である．例を図 5 に示す．このように，クローンの集合のうち 1 つのクローンにだけ変更が加えられると，その部分とクローン関係になる箇所がなくなり，結果としてその部分はコードクローンとならなくなる．そのようなクローンの判定として，以下の 3 つの条件を全て満たしている場合に，条件 2 を満たしていると判断した．

(i) 隣り合うバージョン V_i, V_{i+1} におけるクローンの数を N_i, N_{i+1} とすると，$N_i > N_{i+1}$ である

(ii) バージョン V_i において，コミットがあり，かつ，バージョン V_{i+1} において履歴関係にあるクローンが存在しないようなクローンが 1 つ以上存在する

(iii) (ii) を満たすクローンについて，

(クローンの行数) > (コミットによる削除行数) $*$ 0.30

を満たすクローンが 1 つ以上存在する

(4) 分岐がバグに影響を与えているかを確認するために，**(3)** で得られたクローンセットヒストリーの中の，バグ有り・分岐有りのクローンセットヒストリーについて，さらに，バグの発生時期が分岐の発生時期の前か後かで分類を行った．本研究で

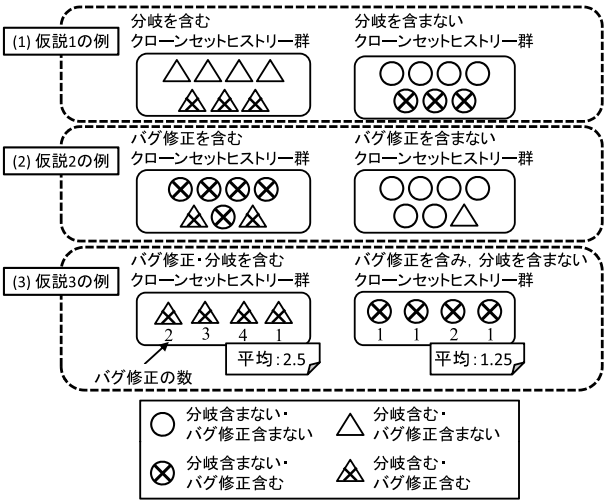

図 3 仮説の説明

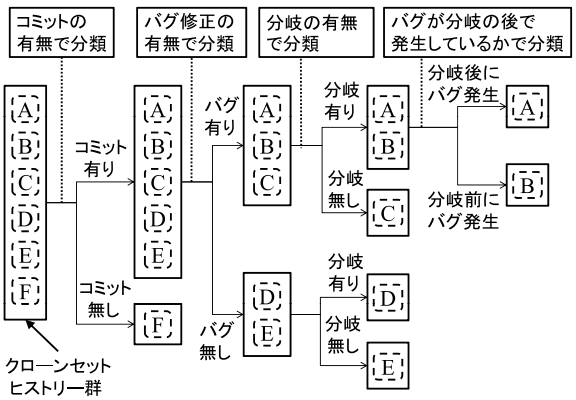

図 4 クローンセットヒストリーの分類手順

は，分類するにあたってまず，バグの発生以前に 1 つでも分岐が存在していれば，そのバグを「分岐後に発生したバグ」，逆に，バグ発生以前に 1 つも分岐がない場合は，そのバグを「分岐前に発生したバグ」としてカウントした．そして,「分岐後に発生したバグ」の数が 1 個以上の場合を「分岐後にバグ有り」，逆に「分岐後に発生したバグ」の数が 0 個の場合を「分岐後にバグ無し」として，クローンセットヒストリーを分類した．

5.2.3 有害性の分析

5.2.2 の手順による分類の結果を用いて有害性の分析を行う．なお，本分析では，分岐がバグの発生に影響するかということについて確認したいため，B グループ「バグ有り・分岐有り・分岐の後にバグ無し」と F グループ「コミット無し」のクローンセットヒストリーは除外した．前節で立てた 3 つの仮説に対するそれぞれの分析についての詳細を以下に記す．

- 分岐の有無によるバグ混入率の比較 (仮説 1 に対する分析)

A，C，D，E 各グループを分岐の有無で分類し，バグを含むクローンセットヒストリーの割合を比較した．分岐有りと分岐無しのそれぞれについて，バグ修正を含むクローンセットヒストリーの割合 p_1, p_2 を計算した (式 1)．

$$p_1 = \frac{|A|}{|A| + |D|} \quad , \qquad p_2 = \frac{|D|}{|C| + |E|} \tag{1}$$

p_1, p_2 について，比率の検定により有意差検定を行った．

- バグの有無による分岐存在率の比較 (仮説 2 に対する分析)

A，C，D，E 各グループをバグの有無で分類し，分岐を含むクローンセットヒストリーの割合を比較した．バグ修正を含む場合と含まない場合のそれぞれについて，分岐が有る割合 p_3, p_4 を計算した (式 2)．

$$p_3 = \frac{|A|}{|A| + |C|} \quad , \qquad p_4 = \frac{|D|}{|D| + |E|} \tag{2}$$

p_3, p_4 について，比率の検定により有意差検定を行った．

- 分岐の有無による平均バグ修正数の比較 (仮説 3 に対する分析)

バグを含むクローンセットヒストリー，すなわち，A，C 各グループについて，バグの有ったクローンの個数の平均を比較した．クローンセットヒストリー A, C のそれぞれについて，バグの有ったクローンの個数を数え，その平均個数を計算した．さらにこの平均個数については，t 検定により，有意差検定を行った．

5.3 結　　果

5.2.2 で示した手順により分類を行った結果と，バグの有るクローンセットヒストリーについては，それぞれに含まれるバグ修正の有ったクローンの数を，表 1 に示す．以上の結果を用いて，5.2.3 で述べた手順に従い，前節で立てた 3 つの仮説を実証するための分析をそれぞれ行った．

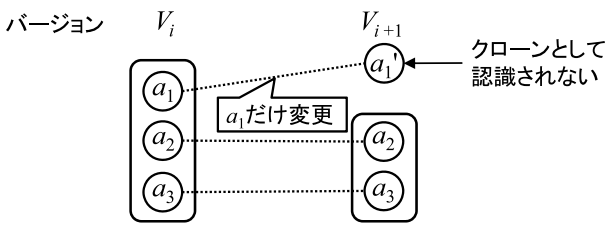

図 5　分岐と判断する場合の例

表 1　クローンセットヒストリーの分類結果

	クローンセットヒストリーの数	バグの有ったクローンの数
A:バグ有り・分岐有り・分岐後にバグ有り	25	179
B:バグ有り・分岐有り・分岐後にバグ無し	4	27
C:バグ有り・分岐無し	156	358
D:バグ無し・分岐有り	109	-
E:バグ無し・分岐無し	1179	-
合計	1473	564

表 2　分岐の有無によるバグ混入率の比較結果

	クローンセットヒストリーの数	バグ有り	バグ有りの割合	検定量 Z_1
分岐有り	134	25	$p_1 = 0.19$	2.35
分岐無し	1335	156	$p_2 = 0.12$	

表 3　バグの有無による分岐存在率の比較結果

	クローンセットヒストリーの数	分岐有り	分岐有りの割合	検定量 Z_2
バグ有り	181	25	$p_3 = 0.14$	2.34
バグ無し	1288	109	$p_4 = 0.08$	

表 4　分岐の有無による平均バグクローンの個数の比較結果

	クローンセットヒストリーの数	バグクローンの個数	平均バグクローンの個数	p 値
分岐有り	25	179	7.1	0.00
分岐無し	156	358	2.3	

（1） 分岐の有無によるバグ混入率の比較

結果を表 2 に示す．バグ有りの割合に関して，有意差検定を行った結果，有意水準 5 % で有意差が確認できた．

（2） バグの有無による分岐存在率の比較

結果を表 3 に示す．分岐有りの割合に関して，有意差検定を行った結果，有意水準 5 % で有意差が確認できた．

（3） 分岐の有無による平均バグ修正数の比較

結果を表 4 に示す．平均個数に関して，有意差検定を行った結果，有意水準 1 % で有意差が確認できた．

6. 考　　察

6.1 クローンの履歴情報を利用した分析

5.3 で示した 3 つの分析結果から，それぞれ，(1) 分岐有りのクローンセットヒストリーの方が，バグを含む割合が高かった，(2) バグ有りのクローンセットヒストリーの方が，分岐を含んでいる割合が高かった，(3) 分岐を含むクローンセットヒストリーの方が，バグの有るクローンを平均的に多く含んでいたことが分かった．

(1)，(2)，(3) は，それぞれ 4. 節で立てた仮説 1，仮説 2，仮

説 3 を支持する結果といえる．よって,「分岐が生じた場合，その後それらのコード片についてバグが発生しやすくなる」ことが確認できた．したがって，このようなクローンは，ソフトウェアにとって有害であり，注意が必要なクローンである可能性が高いといえる．この知見は，クローンに対する保守作業の際に，分岐を含むクローンセットヒストリーに含まれるクローンに注目することで注意すべきクローンを発見しやすくなるという点で有用である．さらに有害なクローンの一例をオープンソースリポジトリのデータから実際に示せたこと，また，クローンの履歴情報に着目することの有用性を確かめられたという点も本研究の成果である．

6.2 分析結果の妥当性

本分析結果は，クローンを多数含み，コミットも多数回行われた，大規模なオープンソース開発を対象としたものである．よって，他の開発事例においても通用する妥当な知見であるといえるが，今後追試を行うことで知見の補強を行うことが重要である．また，分岐以前に生じたバグを分析対象から排除して 3 つの視点からの比較を行うことによって，分岐によるバグへの影響性に関する評価をより正確に行えたといえる．ただし，現状では以下の観点からの議論を今後進めていく必要があると考えられる．

- **分岐発生からバグ発生までの期間:**

分岐がバグの発生に関与しているとするならば，分岐が生じたバージョンの直後，もしくはできるだけ近いバージョンにおいてバグが発生していると考えるのが自然である．しかし，今回の分析では，このような分岐発生とバグ発生の期間との間の関係については評価していない．

- **分岐の数がより多い経路でのバグの発生頻度:**

今回の分析では，分岐中におけるバグ発生の経路や位置，バグ発生に至るまでの分岐の回数を考慮していない．例えば，分岐の回数が少ない経路でバグが発生している可能性が考えられる．

- **バグの発生と分岐との前後関係を決定する上での定義:**

バグ有り・分岐有りのクローンセットヒストリーに関して，バグ発生以前に 1 つでも分岐が有れば,「分岐の後にバグがある」として分類したが，この分類の定義は問題がある可能性がある．例えば，バグ発生以前に分岐が 1 つ有るが，バグ発生以後には分岐が 5 つ有り，かつその後バグが発生していないという場合があったとする．このような場合,「分岐がバグの発生に関与している」とは言い難い．

7. おわりに

本研究では，有害なクローンの判別を支援するために，クローンの分岐とバグの発生との関係性について分析を行った．分析の結果からは，(1) 分岐の有無によるバグ混入率の比較の結果，分岐有りのクローンセットヒストリーの方が，バグを含む割合が高い，(2) バグの有無による分岐存在率の比較の結果，バグ有りのクローンセットヒストリーの方が，分岐を含んでいる割合が高い，(3) 分岐の有無による平均バグ修正数の比較の結果，分岐を含むクローンセットヒストリーの方が，バグクローンを平均的に多く含んでいる，という 3 点を確認した．これらの結果から，分岐とバグ発生との間には関連性があることを確認した．

今回の分析結果は，ソフトウェア開発・保守の際には,「クローンの分岐，すなわち一部のクローンだけを編集することはできるだけ避けること」,「クローンの分岐が避けられない場合，その部分は有害なクローンと言えるので，今後保守の際にはそのような箇所を優先的に見るよう留意すること」が重要であることを示唆している．また，このような処置を補助するような支援ツールも有用であろう．

今後の課題としては，6.2 節で述べた観点からより深い検証を行うことで，分岐とバグの関係性をさらに明確にする必要がある．また，今回の結果は，1 つのプロジェクトの一部のモジュールに対象を限定して分析したものであるので，分析対象の拡大も必要である．

謝　辞

本研究の一部は，文部科学省「次世代 IT 基盤構築のための研究開発」の委託に基づいて行われた．

文　献

[1] C. Kapser and M. Godfrey, “ ‘Cloning Considered Harmful’ Considered Harmful: Patterns of Cloning in Software,” Empirical Software Engineering, pp.645-692, 2008.
[2] M. Kim, V. Sazawal, D. Notkin and G. C. Murphy, “An empirical study of code clone genealogies,” Proc. Joint European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp.187-196, 2005
[3] J. Krinke, “A Study of Consistent and Inconsistent Changes to Code Clones,” Proc. Working Conference on Rverse Engineering, pp.170-178, 2007.
[4] E. Juergens, F. Deissenboeck, B. Hummel and S. Wagner, “Do Code Clones Matter?,” Proc. International Conference on Software Engineering, pp.485-495, 2009.
[5] 左藤裕紀，亀井靖高，上野秀剛，門田暁人，川口真司，名倉正剛，松本健一，飯田元，“クローンの長さとソフトウェア信頼性の関係の分析,” 信学技報，Vol. 108，No. 242，pp. 43-48, 2008.
[6] I. D. Baxter, A. Yahin, L. Moura, M. Sant’Anna and L. Bier, “Clone detection using abstract syntax trees,” Proc. International Conference on Software Maintenance, pp.368-377, 1998.
[7] CCFinderX, http://www.ccfinder.net/index-j.html/
[8] 川口真司, 松下誠, 井上克郎, “版管理システムを用いたクローン履歴分析手法の提案,” 信学論（D）, Vol.J89-D, No.10, pp.2279-2287, 2006.
[9] CVS, http://www.cvshome.org.
[10] Subversion, http://subversion.tigris.org/
[11] M. Fischer, M. Pinzger, and H. Gall, “Populating a release history database from version control and bug tracking systems,” Proc. International Conference on Software Maintenance, pp.23-33, 2003.
[12] T. Zimmermann, R. Premraj and A. Zeller, “Predicting Defects for Eclipse,” Proc. 3rd International Workshop on Predictor Models in Software Engineering, 2007.